CERTIFICATE OF EXPRESS MAILING

EXPRESS MAIL Mailing Label Number EF 260 430 382 US

Date of Deposit December 28, 2000
I hereby certify under 37 CFR 1.10 that this correspondence is being deposited with the United States Postal Service as "Express Mail Post Office to Addressee" with sufficient postage on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.
Name: J. dePaulo
Signature: [signature]

APPLICATION
FOR
UNITED STATES LETTERS PATENT

09751051.122800

TITLE: METHOD AND SYSTEM FOR STREAMING SOFTWARE APPLICATIONS TO A CLIENT

APPLICANTS:

Dan EYLON
Amit RAMON
Yehuda VOLK
Uri RAZ
Shmuel MELAMED

Attorney Docket No. 006032/00021

METHOD AND SYSTEM FOR STREAMING SOFTWARE APPLICATIONS TO A CLIENT

CROSS-REFERENCE TO RELATED APPLICATIONS:

The present application claims the benefit under 35 U.S.C. § 119 of U.S. Provisional Application Serial No. 60/235,535 entitled "Native Streaming Architecture", filed on September 26, 2000, the entire contents of which is hereby expressly incorporated by reference. The present application is also a continuation-in part of U.S. Patent Application Serial No. 09/120,575 entitled "Streaming Modules" and filed on July 22, 1998.

This application is also related to the following pending U.S. Patent applications, the entire contents of which is hereby expressly incorporated by reference: (a) U.S. Patent Application Serial No. 09/237,792 entitled "Link Presentation and Data Transfer" and filed on January 26, 2000; (b) U.S. Provisional Patent Application Serial No. 60/177,736 entitled "Method and Apparatus for Determining Order of Streaming Modules" and filed on January 21, 2000; (c) U.S. Provisional Patent Application Serial No. 60, 177,444 entitled "Method and Apparatus for Improving the User-Perceived System Response Time in Web-based Systems" and filed on January 21, 2000; and (d) U.S. Provisional Patent Application Serial No. 60/207,632 entitled "Apparatus and Method for Improving the Delivery of Software Applications and Associated data in Web-based Systems" and filed in March 25, 2000.

FIELD OF THE INVENTION:

The present invention is directed to a method, system, and architecture for streaming applications from a server for execution on a client.

BACKGROUND:

The Internet, and particularly the world-wide-web, is a rapidly growing network of interconnected computers from which users can access a wide variety of information. Initial widespread use of the Internet was limited to the delivery of static information. A newly developing area of functionality is the delivery and execution of complex software applications via the Internet. There are two basic techniques for software delivery, remote execution and local delivery, e.g., by downloading.

In a remote execution embodiment, a user accesses software which is loaded and executed on a remote server under the control of the user. One simple example is the use of Internet-accessible CGI programs which are executed by Internet servers based on data entered by a client. A more complex system is the Win-to-Net system provided by Menta Software. This system delivers client software to the user which is used to create a Microsoft Windows style application window on the client machine. The client software interacts with an application program executing on the server and displays a window which corresponds to one which would be shown if the application were installed locally. The client software is further configured to direct certain I/O operations, such as printing a file, to the client's system to replicate the "feel" of a locally running application. Other remote-access systems, such as one provided by Citrix Systems, are accessed through a conventional Internet Browser and present the user with a "remote desktop" generated by a host computer which is used to execute the software.

Because the applications are already installed on the server system, remote execution permits the user to access the programs without transferring a large amount of data. However, this type of implementation requires the supported software to be installed on the

server. Thus, the server must utilize an operating system which is suitable for the hosted software. In addition, the server must support separately executing program threads for each user of the hosted software. For complex software packages, the necessary resources can be significant, limiting both the number of concurrent users of the software and the number of separate applications which can be provided.

In a local delivery embodiment, the desired application is packaged and downloaded to the user's computer. Preferably, the applications are delivered and installed as appropriate using automated processes. After installation, the application is executed. Various techniques have been employed to improve the delivery of software, particularly in the automated selection of the proper software components to install and initiation of automatic software downloads. In one technique, an application program is broken into parts at natural division points, such as individual data and library files, class definitions, etc., and each component is specially tagged by the program developer to identify the various program components, specify which components are dependent upon each other, and define the various component sets which are needed for different versions of the application.

Once such tagging format is defined in the Open Software Description ("OSD") specification, jointly submitted to the World Wide Web Consortium by Marimba Incorporated and Microsoft Corporation on August 13, 1999. Defined OSD information can be used by various "push" applications or other software distribution environments, such as Marimba's Castanet product, to automatically trigger downloads of software and ensure that only the needed software components are downloaded in accordance with data describing which software elements a particular version of an application depends on.

Although on-demand local delivery and execution of software using OSD / push techniques is feasible for small programs, such as simple Java applets, for large applications, the download time can be prohibitively long. Thus, while suitable for software maintenance, this delivery system is impractical for providing local application services on-demand because of the potentially long time between when the download begins and the software begins local execution.

Recently, attempts have been made to use streaming technology to deliver software to permit an application to begin executing before it has been completely downloaded. Streaming technology was initially developed to deliver audio and video information in a manner which allowed the information to be output without waiting for the complete data file to download. For example, a full-motion video can be sent from a server to a client as a linear stream of frames instead of a complete video file. As each frame arrives at the client, it can be displayed to create a real-time full-motion video display. However, unlike the linear sequences of data presented in audio and video, the components of a software application may be executed in sequences which vary according to user input and other factors.

To address this issue, as well as other deficiencies in prior data streaming and local software delivery systems, an improved technique of delivering applications to a client for local execution has been developed. This technique is described in parent U.S. Patent Application Serial No. 09/120,575, entitled "Streaming Modules" and filed on July 22, 1998. In a particular embodiment of the "Streaming Modules" system, a computer application is divided into a set of modules, such as the various Java classes and data sets which comprise a Java applet. Once an initial module or modules are delivered to the user, the application begins to execute while additional modules are streamed in the background. The modules are streamed to

the user in an order which is selected to deliver the modules before they are required by the locally executing software. The sequence of streaming can be varied in response to the manner in which the user operates the application to ensure that needed modules are delivered prior to use as often as possible.

Although an improvement over existing streaming technology, the "Streaming Modules" methodology generally operates on software-program boundaries and therefore the streaming is flexibility is constrained to some extent by the structure of the application files and the application itself. In addition, in one embodiment, client-side streaming functionality is added to the streamed program through the use of stub routines inserted into the program code. Thus, the source or object code of the program must be modified to prepare them for streaming.

In a newly developed application streaming methodology, described in co-pending U.S. Patent Application entitled "Method and System for Executing Network Streamed Applications", filed concurrently with the present application, the client system is provided with client-side streaming support software which establishes a virtual file system ("VFS") and connects it to the client's operating system such that the virtual file system appears to be a storage device. The VFS is configured as a sparsely populated file system which appears to the operating system to contain the entire set of application files but, in practice, will typically contain only portions of selected files. Client streaming functionality is provided to process streamlets or blocks of individual files and add them to the VFS as appropriate.

SUMMARY OF THE INVENTION:

The present invention is directed to an improved streaming server method and system configured to stream applications to a client machine running appropriate streaming

support software with client-side execution of the application. The client machines connect to the server over a network, such as the Internet, or another data network, such as a wired or wireless TCP/IP Wide Area Network (WAN). The application need not be fully installed on the client. Instead, the application is streamed to the client in streamlets or blocks which can be stored in a client-side virtual file system configured such that the application can begin to execute on the client machine after only a small fraction of the application is loaded.

More specifically, the application to be executed is stored at the server as a predefined set of blocks or "streamlets" which correspond to specific parts of the various files needed by the application to execute. In a preferred embodiment, each streamlet block corresponds to a file data block which would be loaded by the native operating system as it loads a file while running the application on the client system if the entire application were fully installed at the client. For example, standard Windows systems utilize a 4K code page when loading code blocks from disk or in response to paging requests. Preferably, each streamlet is stored in a pre-compressed format on the server and decompressed upon receipt by the client.

A suitable client system has streaming support software which is configured to provide a sparsely populated virtual file system ("VFS") which appears to the client operating system to be a local drive on which the entire application is stored. However, in practice, only pieces of the various files required for execution of the application may actually be present. During normal operation of an application, the Windows operating system will issue requests to load portions of the application files into memory. Conventional operating system procedures direct these I/O requests to the proper data device driver for processing. When the streaming application is initialized, the operating system is informed that the relevant data is stored on the VFS drive. Thus, as the streaming application executes and paging or data I/O requests are

generated to retrieve require code or data, the operating system will automatically direct these requests to VFS for processing. If the streamlets corresponding to the requested data are present in theVFS, the appropriate data is returned to the operating system and the streaming program continues normal operation. If one or more of the requested streamlet blocks are absent from the VFS, the client issues a fetch request to the server to retrieve the needed code or data.

The server system comprises an application streaming manager which coordinates the transmission of application streamlets to one or more client systems. To improve responsiveness of the application on the client side, the server uses a predictive engine to determine an optimal order to provide streamlets to a client. The predictive engine operates on one or more predictive models which can be generated using information gathered from user interaction with the streaming application and monitoring the order in which various sections of the application's files are accessed as the application runs. Several predictive models can be provided for a given application depending on the various ways in which the application can be used. Alternatively, a single predictive model can be provided with different flows to reflect various types of user behaviors and appropriate weightings between the flows provided. The predictive model can be further modified based on feedback provided by the client system about how an application is being run on a particular client system.

When the server receives a request from a client to start a particular streamed application for the first time, the file structure for the application is forwarded to the client. Preferably, a starting set of streamlets sufficient for enabling the application to start execution is forwarded to the client as well. A program thread to coordinate the pushing of streamlets to a given client is started on the server which actively pushes streamlets to the client in accordance

with the predictive model and possibly additional data forwarded to the server from the client while the application executes.

According to a further aspect of the invention, the server maintains a record of which streamlets have been sent to a client and uses this data as part of the predictive streaming process. The server includes additional functionality to detect when its record of the contents of a client's VFS may be incorrect. In such a situation, the map of the client contents is updated and this information is provided to the predictive streaming engine so that more appropriate decisions can be made regarding which streamlets to send, thus improving the overall network utilization and streaming efficiency.

Advantageously, and unlike remote-access application systems, the streaming applications execute on the client machine while the server and network resources are utilized primarily for data delivery. As a result, server and network load are reduced, allowing for very high scalability of the server-based application delivery service which can deliver, update and distribute applications to a very large number of users, as well as perform centralized version updates, billing, and other management operations. Furthermore, because the application server does not run application code but only delivers streamlets to the client, it can run on operating systems other than the application target operating system, such as Unix derivatives or other future operating systems, as well as Windows-based servers.

The present invention can be used as an enabling technology for remote application hosting, or application service provision (ASP). As broadband technology for connecting end-users to the Internet becomes more widespread, the advantages of application hosting and provision increase. However, since bandwidth limitations are likely to exist for some time, especially when compared to the bandwidth available to a locally executed

application, the present invention provides ways to enhance an application delivery and make it more efficient, while at the same time creating a layer of separation between the Client and the Server, enabling much better control over server integrity and security.

BRIEF DESCRIPTION OF THE FIGURES:

The foregoing and other features of the present invention will be more readily apparent from the following detailed description and drawings of illustrative embodiments of the invention in which:

FIG. 1 is a block diagram of a system for implementing the present invention;

FIG. 2 is a high-level block diagram of the client and server architecture;

FIG. 3 is a sample usage graph used to predict usage of an application program;

FIG. 4 is a block diagram of the server system;

FIG. 5 is shown a high-level flowchart of the operation of a central streaming manager system; and

FIGS. 6A and 6B are flow charts illustrating the operation of one implementation of a streaming management thread.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT(S):

Turning to Fig. 1, there is shown block diagram of a system 10 implementing various aspects of the present invention. The system includes a server 12 which can be accessed by one or more clients 14 via a data network 16, such as the Internet, an intranet, extranet or other TCP/IP based communication network, or other types of data networks, including wireless data networks.

The server 12 contains data for use in streaming one or more software applications to a client on request. To prepare an application for streaming, the various files used for the application are divided into small segments called streamlets which are then stored, preferably in a compressed form, in a suitable library accessible to the server. In addition, a file structure specification for each hosted application which defines how the various files associated with an application appear to a computer when the application is locally installed is provided on the sever. An application's file structure specification can be packaged in a Startup Block which is sent to a client when streaming of the application begins. The Startup Block can also contain startup streamlet set which includes at least those streamlets containing the portions of the application required to enable execution of the application to be initiated, and preferably those streamlets required to begin application execution and have the application run to a point where user interaction is required. In addition, further application information, such as environmental variable settings, additions to system control files, and other system modifications or additions which may be required to "virtually install" the application can be provided. A preferred technique for generating the streamlet sets for an application and determining the contents of the Startup Block is disclosed in U.S. Patent Application entitled "Method and System for Streaming Software Applications to a Client" and filed concurrently with the present application, the entire contents of which is expressly incorporated by reference.

Once the startup streamlet set is received and loaded at the client, the client initiates execution of the application. Concurrently, the server continues to push streamlets to the client in accordance with the predictive model. In addition, the server is responsive to fetch requests issued by clients to retrieve specific streamlets which have not yet been delivered or

were previously delivered and then subsequently discarded by he client. The streamlets can be forwarded to a client individually or grouped together and pushed in clusters as appropriate.

Various embodiments of the client system can be used. In a preferred embodiment, the client system is configured with a virtual file system which appears to the client operating system as a local drive on which all the files needed by the streaming application reside. When a required segment is not present on the client machine, the client issues a request for the appropriate streamlet(s) to the server. Usage information 20 can also be sent from the client 14 to the server 12 and can be used by the server to determine which streamlets to provide next. A preferred client system is disclosed in U.S. Patent Application entitled "Method and System for Executing Network Streamed Applications" and filed concurrently with the present application, the entire contents of which is expressly incorporated by reference.

Fig. 2 is a block diagram of the high-level architecture of the server 12 and client 14. As shown, the server 12 has access to one or more databases which store information used to manage the application streaming. In one embodiment, the server can access an application library 171 which contains the various files for the application. Preferably, the application files are preprocessed and stored as predefined sets of compressed data streamlets each corresponding to a data block in a particular application file starting at a particular offset and having a predefined length, such as the 4k standard Windows operating system file I/O block. In an alternate but less preferred embodiment, the server can generate corresponding streamlets on-the-fly by extracting an appropriate data block from a specified file and offset.

The server also has access to a predictive data model 172 which contains information about the usage patterns of the application, such as the probability that the application will require a given streamlet block Bn if the last block used was Bm. A user

database 173 can also be provided for storing various user-specific information for use in authorizing access to streamed application, storing information related to streaming order, etc. The sever system is discussed in more detail below.

The preferred client system 14 comprises an operating system 140 which can access a variety of local system resources 142, such as data storage devices and memory. A streaming support system module 102 is provided and contains streaming control software which is generally configured to initiate the streaming of streamlets from the server 12 through, e.g., network socket 144, initiate the execution of a specified streaming application 100, process received streamlets, request specific streamlets from the server when necessary, and make received streamlets available for use by the application in a manner which is generally transparent to the native operating system 140.

Preferably, the server 12 is configured to automatically forward sequences of streamlets to the client using a predictive streaming engine which selects streamlets to forward according to the predictive model, such as a dynamic statistical knowledge base generated by analyzing the various sequences in which the application program attempts to load itself into memory as various program features are accessed. As a result, the streamlets 18 which are predicted to be needed at a given point during execution of the application are automatically sent to the client 14 so that they are generally present before the application attempts to access them. Both code and data, including external files used by the application, can be predictively streamed in this manner and, the server is generally unconcerned with the actual content of the streamlets which are sent.

Various statistical and other predictive techniques can be used to analyze the sequence of code and data loads generated by an operating system as it executes an application

and determine an optimal order to push the application streamlets to the client. In one embodiment, the predictive knowledge base can be used which can be viewed as a graph where a node is a user request (e.g. save, load) or a request by the operating system to load a given portion of one the application files, and an edge is the calculated probability that such a request will be made. A simple example of such a graph is shown in Fig. 3. For simplicity, the nodes are illustrated as basic application functions. By examining the links flowing from a given node, the predictive engine can easily determine the most likely future requests and, with access to an appropriate database, determine the streamlets which will be needed by the application to execute those requests. When the user strays from the predictive path and a streamlet fetch request is issued, the predictive routine can be re-centered on a node which corresponds to functionality associated with the requested streamlets and the predictive stream restarted from that point.

In a preferred embodiment, the server updates a prediction model for an application in response to usage data provided by the clients connected to the server. In this manner, the server can adapt its model to different usage patterns, depending on the population of users who actually use the application. Several different models can be provided for each application based on the type of user at issue. For example, one set of users accessing a desktop publishing application may concentrate on text-based functionality while another set of users accesses generally graphical functions. Different predictive models may be appropriate for the various user-types. The system can dynamically assign users to various type categories based on an analysis of their usage patterns. Personalized predictive models can also be generated in which the activity of a particular user is analyzed and used to generate a personalized predictive model.

As will be recognized by those of skill in the art, other predictive analysis techniques for use in selecting which streamlets to forward to the client can also be used, such as Neural Networks and various statistical techniques, and the particular predictive technique uses is not critical to the invention. While the efficiency with which a streamed application is executed on the client machine can vary depending on the order in which the streamlets are delivered, provided that the server is generally compatible with the basic communication protocols and data format used by the client, the operation of client side streaming support software is not generally dependant on the specific manner in which the server system is implemented.

Fig. 4 is a more detailed block diagram of the server system 12 showing various program modules and database systems. In this embodiment, the server has two primary modules: a streaming manager 400 and a prediction engine 402. Various databases, archives, etc. are also provided. The streaming manager 400 is responsible for coordinating the streaming of applications to one or more clients 14.1 – 14.N. The prediction engine 402 is accessed by the streaming manager 400 and identifies the next one or more streamlets which are predicted to be most appropriate to send to a given client at a given time.

The streaming manager 400 can be connected to the communication network 16 in a variety of ways. In general, access will be provided through a suitable I/O module 414 connected to the network. Depending on the manner of implementation, a streamlet queue 412 can be provided in which the streaming manager 400 can place streamlets directed to a particular client for subsequent transmission.

As discussed above, the server is configured to stream blocks or sections of application files to the client and the streamlets for the various files of an application can be

stored in the application library 171. The stored data can be organized in many ways. In a preferred embodiment, each streamlet corresponds to a compressed data block from a particular file and is stored using the file name and offset of the block's starting location in that file as an index in the database. When a streamlet is forwarded to the client, the transmission indicates the file and offset which define the source of the streamlet data, either directly or via a unique name which can be referenced to a file and offset lookup table by the client. Error detection and recovery information can also be included in the transmission.

The application library 171 can also provide storage for the Startup Block which is forwarded to the client when a streaming application is first started. The block can be preassembled. Alternatively, the various components of the block can be combined by the streaming manager when appropriate such that the Startup Block does not contain elements which are already on the client system.

The application library 171 can also store one or more prediction models for a given application. The models can be subsequently accessed by the prediction engine 402 directly from the application library database 171. Alternatively, the prediction models can be stored in a separate library or database 404 in a form which may be more suitable for use by the prediction engine 402. For example, the application library 171 can contain the master set of prediction models while the default model library 404 is populated by prediction modules extracted from the library 171 when a streaming application is started. The modules in library 404 can subsequently be altered during an active streaming session, i.e., in response to usage behavior of one or more users forwarded by the client systems, without fear of corrupting the mater copy. Periodically, such updated prediction models can be returned to the application library 171.

Alternatively, or in conjunction, the prediction engine can operate on a default prediction model as modified by a differential model 406. A differential prediction model can be used to alter various aspects of the basic model based upon information related to a given user or user-type. For example, as a user interacts with a streamed application, information regarding the manner of interaction and usage behavior can be forwarded from the client to the server. Rather than modifying the primary or default prediction model for the streaming application, a personalized differential prediction model for that user can be generated. The differential model can be stored in the user database 173. When a subsequent streaming session is initiated, the differential prediction model associated with the given user can be extracted from the user database and provided to the prediction engine so that the default prediction model for the streaming application will be modified in a manner best suited for the given user. Depending on the manner in which the prediction models are implemented, it may be more appropriate to store a complete prediction model for each user which can be freely modified rather than storing variations or differentials of the default model.

As discussed above, in order to process streaming applications, a suitable client environment, such as the virtual file system discussed above, should be provided on the client. To this end, a streaming environment software repository 408 can be made available to the server. Repository 408 can be used to store different versions of client-side software for use on various client computing platforms. When a client first requests use of a streaming application, suitable environmental software can be retrieved from the repository 408 and installed on the client system. Suitable automated installation processes will be known to those of skill in the art.

During the streaming process, the streaming manager 400 can maintain a record of the status of the various clients and the applications being streamed to them. This information

can include, for example, a given user's position in the prediction model for a streaming application. According to one particular aspect of the invention, the streaming manager 400 can also maintain a record of the streamlets which have been sent to a given client. In a preferred embodiment, the record is maintained as a bit map with each bit corresponding to a particular streamlet. This record can be used, e.g., by the prediction engine 402, to determine appropriate streamlets to send to a client at a given time since an initial while avoiding resending streamlets which are already present, thus improving the overall network utilization and streaming efficiency.

Under some circumstances, the server-maintained image of the client's contents can become out of sync. For example, the client may discard data blocks from its VFS to make room for new streamlets. In this case, eventually, an on-demand request for a particular streamlet will be issued by the client for data which the server shows as already being present on the client side. The Server can then request an image update, in response to which the Client will send a current image of the VFS population, preferably in the same format as the server map. The client can also send an image to the server when a previously run application is being restarted or in response to the server sending a streamlet which is already present on the client to ensure that the server has the most current VFS image.

The software implementation of the streaming manager 400 can be configured in a variety of ways. Figs. 5, 6A and 6B are flowcharts illustrating one method of implementing the server functionality. In general, this streaming software has two primary components. One component is the central system which manages the initiation of a streaming application session, shown in the flowchart of Fig. 5. The second component is the functionality that manages the

actual predictive and interactive streaming session of an application for a given client, shown in the flowcharts of Figs. 6A and 6B.

The prediction engine itself is a third component of the server system. However, the internal functionality of the prediction engine is not the focus of the present invention. Rather, the prediction engine can be implemented using one of many techniques known to those of skill in the art.

Turning to Fig. 5, there is shown a high-level flowchart of the central streaming manager system which manages the initiation of a streaming application session. The streaming process begins when a user requests to start or resume execution of a streaming application. (Step 500). After a suitable logon or user verification process, details regarding the streaming application and the user are retrieved from the appropriate database (step 502). Such data can include, for example, records of prior uses of the application by the user, limits on various features of the application which can be accessed by that user, billing information, etc. In addition, the streaming application's prediction model (and any user-specific versions thereof) are retrieved, if necessary, and made available to the prediction engine.

If the user is a new user of the streaming system (step 504), the type of client system is determined and appropriate streaming environment software is installed on the client system (step 506). After the streaming environment is set up on the client's system, an empty streamlet or VFS map for that particular client is created on the server. (Step 508). An empty VFS map is also created if the user is a returning user but is not restarting a previously streamed application. (Step 504, 510).

If the client is restarting a previously streamed application, for example, after suspending a streaming session for a period of time, the server can retrieve a saved copy of its

local VFS map for that client. However, the contents of the VFS on the client may have changed during the streaming suspension period. Thus, preferably, the server sends a request to the client system asking it to send a copy of its VFS map. This data will indicate which streamlets for the selected application are present on the client's system when the streaming application is restarted. When the map is returned, it is stored on the server, for example, in the application status database 410. (Step 514).

Once the empty VFS map is created or the current VFS map is retrieved from the client, the prediction engine is positioned at the appropriate place in the prediction model for the application. For a new application streaming session, the prediction engine can be located at a point where it will direct that the Init block or Startup block be forwarded to the client. (Alternatively, transmission of the Startup block can be managed by the streaming manager without use of the prediction engine, and the prediction engine only utilized after the initial data transmission process.) If the client has already used the application, and as will be indicated in the VFS map, the client may already have portions of the application stored locally and therefore a different starting position in the streaming prediction model may be warranted. The appropriate position in the prediction module use for streaming can be determined by the streaming manager or a suitable functionality to make this decision can be included in the prediction engine itself. (Step 516).

Finally, a program thread or stream is initiated to manage the particulars for streaming the selected application to that particular client. (Step 518). Preferably a separate streaming thread is initiated on the server for each streamed application being run by each client. Different implementations are also possible and alternative functionality can be provided which will manage multiple application streams simultaneously.

Figures 6A and 6B are flow charts illustrating the functionality which is present in one implementation of a streaming management thread. Four separate program paths are illustrated beginning with steps 600, 620, 640 and 660 respectively. The tasks will generally operate in parallel. Suitable techniques for implementing such a multifunction program thread (or threads) are known to those of skill in the art.

Step 600 is the beginning of the main streaming program loop. Once a streaming application has started, the prediction engine is queried to determine the next streamlet or streamlets which are appropriate to send to the client's system. (Step 600). The appropriate streamlets are then retrieved from the application library (step 602) and transmitted to the client. (Step 604). If the transmission is not successful (step 606) the streamlet can be resent. Upon a successful transmission, the server-maintained copy of the client VFS map is updated to reflect the fact that the sent streamlets are now present on the client system. The process then repeats, wherein the prediction engine indicates the next sets of streamlets to send to the client.

During operation of the application on the client system, the program may need to retrieve portions of the application files which have not yet been provided by the server. When this occurs, the client issues a query to the server requesting that the appropriate streamlets be provided. When such a streamlet request is received from a client (Step 620), the server determines whether the server-maintained VFS map indicates that the requested streamlet has previously been sent to the client. (Step 622). If the requested streamlet has not previously been sent to the client, the streamlet is retrieved from the applications library. (Step 624). In addition, the position in the prediction engine may need to be repositioned within the prediction model to account for the fact that the user has place the application in a state which requires streamlets

that were not predictively sent. (Step 626). The retrieved streamlet is then forwarded to the client and the server maintained VFS map is updated accordingly. (Step 628, 630, 632).

In some cases, the server maintained VFS map will indicate that the streamlet requested by the client is already present on the client's system. This can occur, for example, when the client has purged streamlets from memory or disc to make room for more frequently or more recently required streamlets. In response to such a mismatch (or after a predetermined number of such mismatches have been detected), the server requests the current VFS map from the client. (Step 634). After the client's VFS map is received (step 636) the server's copy is replaced with the map returned from the client. (Step 638). The requested streamlet is then retrieved from the application library, the prediction engine repositioned if needed, and the streamlet sent to the client. (Steps 624 - 632). It should be noted that the client's VFS map could be requested after the streamlet has been sent or the streamlet transmission and VFS synchronization can be performed in parallel.

There are also situations when the client's system can detect that the server may not have an up-to-date version of the VFS map or that the server map may have been corrupted. For example, the server can send a streamlet to the client which the client already has. When such a mismatch is detected, the client can indicate this synchronization error by sending a current copy of its VFS map to the server. As shown in Fig. 6B, when the server receives a current VFS map from a given client (Step 640) it assumes that the client has detected such mismatched condition and replaces the local map with the updated map received from the client. (Step 642). The prediction engine can also be repositioned based on the differences between the old and new VFS maps. (Step 644). The server and client-supplied maps can be compared (e.g., by applying an XOR function) to identify and log the mismatch and this later analyzed to

determine the cause of the unnecessary streamlet transmission, such as an error in the prediction engine.

In addition, and as discussed above, during the course of executing the streamed application on the user's system, the client can periodically send user activity summaries, such as application usage tracking information, to the server indicating the order in which various parts of the application file have been accessed by the user. When a user activity summary is received at the server (Step 660) it can be used to update the various predictive models for the streaming application. (Step 662). For example, the application usage data can be used to generate or modify the differential prediction model 406 associated with the particular client or client type and which is used in conjunction with a default prediction model to determine which streamlets to send to the client at a given time. User activity data can also be analyzed in the aggregate and used to update the default prediction model as required. Various other uses for such data are also possible.

According to further aspects of the invention, the general application streaming methodology disclosed above can be integrated with a billing systems to implement various Application Service Provider models, such as Pay-Per-Use, time-based payment, subscriptions, or function-based payment, etc. The management of this functionality can be provided by a billing module (not shown) which interfaces with the streaming manager 400 to monitor usage of streamed applications by the various users, indicate usage limitations, record usage charges, etc.

Various hardware platforms can be used to implement the server system. Preferably, the streaming server is implemented using a Windows NT or Solaris environment and is connected to the Internet using conventional means. As will be appreciated, because the server does not execute the streamed program, but simply pushes data blocks and services

requests, the server's operating environment does not need to be compatible with the client environment.

While the invention has been particularly shown and described with reference to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the spirit and scope of the invention. For example, different applications may reside on multiple servers and a user executing several streaming applications may access multiple servers. In addition, the breakdown and division of functionality between the various software routines and databases in the server 12 can vary depending on implementation issues and design choices such that the functionality attributed to one module can be performed by multiple software routines and the functionality of two or more modules can be combined into a single routine.